

Main.c

```

/*****
 *
 *****/
 *
 * PROJECT:          e-sistant for action-engine
 *
 * FILE:             Main.c (Phase 1)
 *
 * DESCRIPTION:
 *
 *  OBJECTIVE:
 *
 * REVISION HISTORY:
 *   5/1/99 Brian Roundtree    Initial version
 *
 *****/

#include <Pilot.h>           // all the system toolbox headers
#include <PalmOS.h>          // NEW for 3.5, all the system toolbox headers
#include <PalmCompatibility.h>
#include "CharAttr.h"
#include "Chars.h"           // special, the over the air symbol
#include "SystemMgr3.h"      // special hacks for 3.1
#include "Globals.h"

#include "osAddressDB.h"
#include "osDatebook.h"
#include "osDbMaker.h"

#include "AddressDB.h"
#include "Conversant.h"
#include "Parser.h"
#include "Main.h"
#include "Main_res.h"        // application resource defines

/*****
 *
 * Entry Points
 *
 *****/
DWord PilotMain (Word cmd, Ptr cmdPBP, Word launchFlags);
static DWord IslandMain (Word cmd, Ptr cmdPBP, Word launchFlags);

/*****
 * Global variables for this module
 *****/
static MenuBarPtr    CurrentMenu = NULL;    // ptr to current menu
static MatchList     matchedItems;         // Possible list item matches found
static Sentence      currentPlan;          // Plan structure
static DmOpenRef     gListDB;              // Global database of ESIS list, kept open while running
static UnPkEsisRecd  gUnpkdDbRecd;         // Record holder for unpacked list currently in use

/*****
 * Prototypes for internal functions
 *****/
static void LoadUrlInfo(Ptr cmdPBP);
static void StartApplication(void);
static void StopApplication(void);

```

Main.c

```

static void SetCurrentMenu(Word rscID);
static Err GoToURL(CharPtr origurl);
static Boolean GetPlanUrlFormat(CharPtr str);
static Boolean ApplicationHandleEvent(EventPtr event);
static void EventLoop(void);
static void MakeEsisList(UnPkEsisRecdPtr esisRecd, UInt listObj);
static Err OpenOrCreateDB(DmOpenRef *dbP, ULong type, ULong creator,
    ULong mode, char *name, Boolean *created);
static void InitializeListDB(void);
static void HandleEsisFieldProcessing ( EventType *event);
static Err CheckOtherNeededDatabases (void);
static void HandlePickFromList(UInt pick);

/***** Routines Called by the System *****/
static void EsisListDrawFunc(Word itemNum, RectanglePtr bounds, CharPtr *s);
static void DrawCharsToFitWidth(const char *s, RectanglePtr r);

/*****
 *
 * FUNCTION:   StartApplication
 *
 * DESCRIPTION: This routine sets up the initial state of the application.
 *
 * PARAMETERS: None.
 *
 * RETURNED:   Nothing.
 *****/
static void StartApplication(void)
{
    FormPtr      frm;
    FieldPtr     fldP;
    UInt         mode = dmModeReadWrite;
    Boolean       created;
    Err           error;

    frm = FrmInitForm(MainForm);
    FrmSetActiveForm(frm);

    SetCurrentMenu(MainMenuBar);

    OpenOrCreateDB( &gListDB, kListDBType, kEsisCreator, mode,
        kListDBName, &created); // Find the LIST database. If it doesn't exist, create it.
    if (created)
        if (!(error = CheckOtherNeededDatabases()))
            ErrFatalDisplayIf(error, "Could not find Database. [BuildSentenceFromStruct]");
    gUnpkdDbRecd.next[0].type = kTypeAction;
    GetListByTypeEsis(kTypeAction, gListDB, &gUnpkdDbRecd, kStartNew);
    MakeEsisList(&gUnpkdDbRecd, MainPickListList); // Load a default starting list

    fldP = FrmGetObjectPtr(frm, FrmGetObjectIndex(frm, MainTextLineField));
    FldSetInsPtPosition (fldP, 0);
    FrmDrawForm(frm);

    LoadTalkField(""); // Init the field
    SysTaskDelay(SysTicksPerSecond());
}

```

Main.c

```
LoadTalkField("What can I do for you?");           // finish the greeting

FrmSetFocus (frm, FrmGetObjectIndex(frm, MainTextLineField));
}

/*****
 *
 * FUNCTION:  TerminateApplication
 *
 * DESCRIPTION: This routine cleans up the application for quitting
 *
 * PARAMETERS: None.
 *
 * RETURNED:  Nothing.
 *****/
static void StopApplication(void)
{
    if(gListDB) { // error check the pointer, if we switchUI(goToURL) the this global is bad
        DmCloseDatabase(gListDB); // this should be the only open DB
        gListDB = NULL; // clears it so we can test for NULL on a second pass
    }
}

/*****
 *
 * FUNCTION:  SetCurrentMenu
 *
 * DESCRIPTION:
 *
 * PARAMETERS: rscID - resource id of the new menu
 *
 * RETURNED:  nothing
 *****/
static void SetCurrentMenu(Word rscID)
{
    if (CurrentMenu)
        MenuDispose(CurrentMenu);

    CurrentMenu = MenuInit(rscID);
}

/*****
 *
 * FUNCTION:  GoToURL
 *
 * DESCRIPTION:
 *
 * PARAMETERS: parameter is ptr to URL string
 *
 * RETURNED:  error, 0 means no error
 *****/
Err GoToURL(CharPtr origurl)
{
```

Main.c

```
Err err;
CharPtr url;
DmSearchStateType searchState;
UInt cardNo;
LocalID dbID;
long strLen;

StopApplication(); // cleanup because we loose globals after we call switchUI

strLen = StrLen(origurl); // make a copy of the URL, since the OS will free
url = MemPtrNew(strLen+1); // the parameter once Clipper quits
if (!url)
    return sysErrNoFreeRAM;
StrCopy(url, origurl);
MemPtrSetOwner(url, 0); // find clipper and launch it

err = DmGetNextDatabaseByTypeCreator (true, &searchState,
    sysFileTApplication, sysFileCClipper, true, &cardNo, &dbID);
if (err) { // Clipper is not present
    FrmAlert(NoClipperAlert);
    MemPtrFree(url);
}
else
    err = SysUIAppSwitch(cardNo,dbID,sysAppLaunchCmdGoToURL,url);
return err;
}

/*****
*
* FUNCTION:      ApplicationHandleEvent
*
* DESCRIPTION:   Handles processing of events for the 'main' form.
*
* PARAMETERS:    event      - the most recent event.
*
* RETURNED:      True if the event is handled, false otherwise.
*
*****/
static Boolean ApplicationHandleEvent(EventPtr event)
{
    Boolean    handled = false;
    FormPtr    frm;

    switch (event->eType) {
    case ctlSelectEvent:
        if (event->data.ctlSelect.controlID == MainHandleItButton) {
            Char    strGetUrl[kMaxUrlSegment];
            currentPlan.plan.userID = TimGetSeconds(); // gives the transaction a unique ID
            if (GetPlanUrlFormat((CharPtr)&strGetUrl)) {
                GoToURL((CharPtr)&strGetUrl);
                handled = true;
            }
        }
        else
            handled = false;
    }
    else if (event->data.ctlSelect.controlID == MainRememberPswdCheckbox) {
        handled = false;
    }
}
```

Main.c

```

    }
    break;
case lstSelectEvent:
    if (event->data.lstSelect.listID == MainPickListList ) {
        HandlePickFromList( event->data.lstSelect.selection);
        handled = false;
    }
    break;
case penUpEvent:
    handled = false;
    break;
case menuEvent:
    MenuEraseStatus(CurrentMenu);
    frm = FrmInitForm(AboutForm);
    FrmDoDialog(frm);
    FrmDeleteForm(frm);
    handled = true;
    break;
default:
    break;
}
return handled;
}

```

```

/*****
*
* FUNCTION:      GetPlanUrlFormat
*
* DESCRIPTION:   Use the global currentPlan to create a formatted
*                data URL & string
*
* PARAMETERS:
*
* RETURNED:     true if a good string to send.
*
*****/
static Boolean GetPlanUrlFormat(CharPtr str)
{
    char s[42];
    UInt i;
    ULong temp;

    //StrCopy((CharPtr)&strGetUrl, "file:action.pqa/");

    //Standard header
    StrCopy(str,"http://207.202.236.19/scripts/action.dll?Schedule?l=[da.a]&Z=[za]");
    // StrCopy(str,"http://207.202.236.19/scripts/action.dll?ParmTest?l=[da.a]&Z=[za]"); //Test the device ID codes
    // StrCopy(str,"http://207.202.236.19/scripts/action.dll?ParmTest?l=%DEVICEID&Z=%ZIPCODE"); //Test the
    device ID codes

    // T=Type of Data param to follow, it is converted into type long
    StrCat(str, "&T=");
    temp = 1;
    StrPrintf(s,"%Ld",temp);
    StrCat(str, (CharPtr)&s);

    // U=Unique user planID, this number is assigned by the users device (Palm is seconds since 1904)

```

Main.c

```

// when the plan done and the user presses Handle it! or similar function. The time is converted to
// HEX to keep the string shorter (it saves a few bytes)
StrCat(str, "&U=");
StrPrintf(s, "%Lx", currentPlan.plan.userID);
StrCat(str, (CharPtr)&s);

// P=Param, what who where when, 4 byte type code ie-'ACTN' + 2 byte in Hex ie-'1C03' leave off the "0x" part
// Should look like "ACTN0001NAME00000003PLCL00000003WHEN0004" 8 bytes long. These are linked with
// no spaces. Use as many as needed.
StrCat(str, "&P=");
for (i = 0; i < currentPlan.tree.blkCount; i++) {
    ReadRecdPtrLengthToPtr( (UCharPtr)&s, (UCharPtr)&(currentPlan.tree.block[i].sType), 4);
    s[4] = '\0'; // terminate it
    StrCat(str, (CharPtr)&s);
    switch (currentPlan.tree.block[i].sType) {
        case kTypeTalk:
            StrPrintf(s, "%x", 0); // _DEBUG
            break;
        case kTypeList:
            StrPrintf(s, "%x", 0); // _DEBUG
            break;
        case kTypeAction:
            StrPrintf(s, "%x", currentPlan.tree.block[i].sListItem);
            break;
        case kTypeName:
            StrPrintf(s, "%Lx", currentPlan.tree.block[i].sdbRecUniqueID);
            break;
        case kTypeNamePalm:
            // _DEBUG below for demo only VVVVV
            //GetNameByUniqueID((CharPtr)&s, &gUnpkdDbRecd, currentPlan.tree.block[i].sdbRecUniqueID, 0 );
            StrCopy(s, "John");
            break;
            // _DEBUG above for demo only ^^^^^
            //This is the real code VVVVV
            StrPrintf(s, "%Lx", currentPlan.tree.block[i].sdbRecUniqueID);
            break;
        case kTypePlace:
            StrPrintf(s, "%Lx", currentPlan.tree.block[i].sdbRecUniqueID);
            break;
        case kTypeTime:
            StrPrintf(s, "%x", 0); // _DEBUG
            break;
        case kTypeWhere:
            StrPrintf(s, "%x", 0); // _DEBUG
            break;
        case kTypeWhen:
            StrPrintf(s, "%x", currentPlan.tree.block[i].sListItem);
            break;
        case kTypeNameLocation:
            StrPrintf(s, "%x", currentPlan.tree.block[i].sListItem);
            break;
        default:
            StrCopy((CharPtr)&s, "FFFF"); // Error
            return false;
    }
}

```

```

        StrCat(str, (CharPtr)&s);
    }
    return true;
}

/*****
 *
 * FUNCTION:      HandlePickFromList
 *
 * DESCRIPTION:
 *
 * PARAMETERS:
 *
 * RETURNED:      Nothing.
 *****/
static void HandlePickFromList(UInt pick)
{
    LoadSentenceRecord( &gUnpkdDbRecd, &currentPlan, gListDB, pick );
    ParseUpdateList(0, MainPickListList, NoSelectItem); // reset the list selection to the top
}

/*****
 *
 * FUNCTION:      EventLoop
 *
 * DESCRIPTION:
 *
 * PARAMETERS:      None.
 *
 * RETURNED:      Nothing.
 *****/
static void EventLoop(void)
{
    EventType    event;
    Word         error;

    do {
        EvtGetEvent(&event, evtWaitForever);
        if (! SysHandleEvent(&event))
            if (! MenuHandleEvent(CurrentMenu, &event, &error))
                if (! ApplicationHandleEvent(&event))
                    FrmHandleEvent(FrmGetActiveForm(), &event);

        if (event.eType == keyDownEvent) { // This is Application handled,
            HandleEsisFieldProcessing( &event); // do it after the field is updated
        }
    } while (event.eType != appStopEvent);
}

/*****
 *
 * FUNCTION:      PilotMain
 *****/

```

Main.c

```

*
* DESCRIPTION:      This function is the equivalent of a main() function
*                   under standard 'C'. It is called by the Emulator to begin
*                   execution of this application.
*
* PARAMETERS:      cmd - command specifying how to launch the application.
*                   cmdPBP - parameter block for the command.
*                   launchFlags - flags used to configure the launch.
*
* RETURNED:        Any applicable error code.
*
*****/
DWord PilotMain(Word cmd, Ptr cmdPBP, Word launchFlags)
{
    return IslandMain( cmd, cmdPBP, launchFlags);
}

/*****
*
* FUNCTION:        IslandMain
*
* DESCRIPTION:
*
* PARAMETERS:
*
* RETURNED:        void
*
*****/
DWord IslandMain(Word cmd, Ptr cmdPBP, Word launchFlags)
{
    Err error = 0;
    Boolean appIsActive = launchFlags & sysAppLaunchFlagSubCall;

    switch (cmd) {
    case sysAppLaunchCmdNormalLaunch:
        StartApplication();
        if (cmdPBP != NULL && StrStr((CharPtr)cmdPBP, kLaunchEsistantAppl))
            LoadUrlInfo(cmdPBP+15);    // offset past the "palm:ESIS.appl?"
        EventLoop();
        StopApplication();
        break;

    case sysAppLaunchCmdURLParams:
        {
            //launch new process
            Boolean launched;

            launched = launchFlags & sysAppLaunchFlagNewGlobals;
            if (launched) {
                StartApplication ();
                if (error)
                    return (error);
            }
            LoadUrlInfo(cmdPBP+19);    // offset past the "palmcall:ESIS.appl\?"
            if (launched) {
                //EventLoop();
            }
        }
    }
}

```


Main.c

```

        StopApplication();
    }
    }
    break;
}
return 0;
}

/*****
 *
 * FUNCTION:      LoadUrlInfo
 *
 * DESCRIPTION:
 *
 * PARAMETERS:
 *
 * RETURNED:      void
 *****/
void LoadUrlInfo(Ptr cmdPBP)
{
    Word ret;

    ret = FrmCustomAlert (EnterSceduleAlert,
                          "I have placed everything in your calendar.",
                          "", "");
    if( ret == EnterSceduleOK ) {
        ParseUrlForCommands ( (CharPtr)cmdPBP); // load the data in the back ground
        //launch the databook and show the date
    }
    else if (ret == EnterSceduleHoldon ) {
        ret = FrmAlert (SchedDoltLaterAlert);
        if (SchedDoltLaterOK)
            ret = _DEBUG1; // NEEDS WORK
        else if (SchedDoltLaterForgetit)
            ret = _DEBUG1; // NEEDS WORK
    }
}

/*****
 *
 * FUNCTION:      MakeEsisList
 *
 * DESCRIPTION:    Builds a linked list and loads it into the 'listObj'.
 *
 * PARAMETERS:    esisRecd ptr to UnPkEsisRecd variable
 *                listObj is the list object id
 *
 * RETURNED:      void
 *****/
static void MakeEsisList(UnPkEsisRecdPtr esisRecd, UInt listObj)
{
    FormPtr    frmP;
    ListPtr    listP;
    frmP = FrmGetActiveForm();

```

Main.c

```

if (esisRecd->listCount) { // do if any items in the list
    listP = FrmGetObjectPtr(frmP, FrmGetObjectIndex(frmP, listObj));
    LstEraseList(listP);

    LstSetListChoices(listP, NULL, esisRecd->listCount);

    LstSetDrawFunction(listP, EsisListDrawFunc);
    LstSetTopItem(listP, 0);
    LstSetSelection(listP, -1);
    LstDrawList(listP);
}
FrmDrawForm(frmP);
}

/*****
*
* FUNCTION:      EsisListDrawFunc
*
* DESCRIPTION:   Used by the List Manager to draw my list
*
* PARAMETERS:    Must use global static UnPkEsisRecd  dbRecd
*
* RETURNED:      void - Do not change the prototype!
*
*****/
static void EsisListDrawFunc(Word itemNum, RectanglePtr bounds, CharPtr *s)
{
    WinDrawChars( (CharPtr)&(gUnpkdDbRecd.list[itemNum].text),
                  StrLen((CharPtr)&(gUnpkdDbRecd.list[itemNum].text)),
                  bounds->topLeft.x,
                  bounds->topLeft.y );
}

/*****
*
* FUNCTION:      OpenOrCreateDB
*
* DESCRIPTION:   open a database. If it doesn't exist, create it.
*
* PARAMETERS:
*
* RETURNED:      Db open error
*
*****/
static Err OpenOrCreateDB( DmOpenRef  *dbP,
                          ULong      type,
                          ULong      creator,
                          ULong      mode,
                          Char       *name,
                          Boolean     *created)
{
    Err  err;

    *created = false;
    *dbP = DmOpenDatabaseByTypeCreator(type, creator, mode);
    err = DmGetLastErr();
}

```

Main.c

```

if (! *dbP) {
    err = DmCreateDatabase(0, name, creator, type, false);
    if (err)
        return err;
    *created = true;

    *dbP = DmOpenDatabaseByTypeCreator(type, creator, mode);
    if (! *dbP)
        return DmGetLastError();
}

if (*created) {
    InitializeListDB();
}

return err;
}

```

/** TEMP ?? */

```

*
* FUNCTION:      InitializeListDB
*
* DESCRIPTION:   initialize a basic database
*
* PARAMETERS:    void
*
* RETURNED:      void
*
*****

```

static void InitializeListDB(void)

```

{
    UnPkEsisRecd  c1, c2, c3;
    TalkList      t1;
    CharPtr       sP;
    VoidHand      h2;
    UnPkEsisRecd  *nDbRec[3];

    UInt          numLists = 3;
    UInt          i, ii;

    c1.id          = 1;
    c1.type        = kTypeAction;
    c1.creator      = kEsisCreator;
    c1.version      = 1;
    c1.priorCount   = 2;
    c1.priorOffset  = 0;
    c1.nextCount    = 2;
    c1.nextOffset   = 0;
    c1.listCount    = 4;
    c1.list[0].uniqueID = 0;
    c1.list[0].esisID  = kEsisIdMask;
    StrCopy((CharPtr)c1.list[0].text, "Arrange meeting w/");
    c1.list[1].uniqueID = 0;
    c1.list[1].esisID  = kEsisIdMask;
    StrCopy((CharPtr)c1.list[1].text, "Take care of ?");
}

```

Main.c

```

c1.list[2].uniqueID = 0;
c1.list[2].esisID = kEsisIdMask;
StrCopy((CharPtr)c1.list[2].text, "Lunch w/");
c1.list[3].uniqueID = 0;
c1.list[3].esisID = kEsisIdMask;
StrCopy((CharPtr)c1.list[3].text, "Dinner w/");
c1.prior[0].reserved = 0;
c1.prior[0].type = 'XXXX';
c1.prior[1].reserved = 0;
c1.prior[1].type = 'XXXX';
c1.next[0].reserved = 0;
c1.next[0].type = kTypeName;
c1.next[1].reserved = 0;
c1.next[1].type = 'XXXX';

c2.id = 2;
c2.type = kTypePlace;
c2.creator = kEsisCreator;
c2.version = 1;
c2.priorCount = 2;
c2.priorOffset = 0;
c2.nextCount = 2;
c2.nextOffset = 0;
c2.listCount = 4;
c2.list[0].uniqueID = 111;
c2.list[0].esisID = kEsisIdMask;
StrCopy((CharPtr)c2.list[0].text, "Home");
c2.list[1].uniqueID = 222;
c2.list[1].esisID = kEsisIdMask;
StrCopy((CharPtr)c2.list[1].text, "My Office");
c2.list[2].uniqueID = 333;
c2.list[2].esisID = kEsisIdMask;
StrCopy((CharPtr)c2.list[2].text, "Lunch");
c2.list[3].uniqueID = 444;
c2.list[3].esisID = kEsisIdMask;
StrCopy((CharPtr)c2.list[3].text, "Dinner");
c2.prior[0].reserved = 0;
c2.prior[0].type = kTypeTime;
c2.prior[1].reserved = 0;
c2.prior[1].type = kTypeWhere;
c2.next[0].reserved = 0;
c2.next[0].type = kTypeWhen;
c2.next[1].reserved = 0;
c2.next[1].type = kTypeWhen;

c3.id = 4;
c3.type = kTypeWhen;
c3.creator = kEsisCreator;
c3.version = 1;
c3.priorCount = 2;
c3.priorOffset = 0;
c3.nextCount = 2;
c3.nextOffset = 0;
c3.listCount = 4;
c3.list[0].uniqueID = 111;
c3.list[0].esisID = kEsisIdMask;
StrCopy((CharPtr)c3.list[0].text, "Pick a day...?");
c3.list[1].uniqueID = 222;

```

```

c3.list[1].esisID      = kEsisIDMask;
StrCopy((CharPtr)c3.list[1].text, "Show possible dates.");
c3.list[2].uniqueID    = 333;
c3.list[2].esisID      = kEsisIDMask;
StrCopy((CharPtr)c3.list[2].text, "ASAP!");
c3.list[3].uniqueID    = 444;
c3.list[3].esisID      = kEsisIDMask;
StrCopy((CharPtr)c3.list[3].text, "Next Week sometime.");
c3.prior[0].reserved   = 0;
c3.prior[0].type       = kTypePlace;
c3.prior[1].reserved   = 0;
c3.prior[1].type       = kTypeAction;
c3.next[0].reserved    = 0;
c3.next[0].type        = kTypeEnd;
c3.next[1].reserved    = 0;
c2.next[1].type        = kTypeEnd;

nDbRec[0] = &c1;
nDbRec[1] = &c2;
nDbRec[2] = &c3;
for (i = 0; i < numLists; i++) {
    UInt index = dmMaxRecordIndex;
    VoidHand h = DmNewRecord(gListDB, &index, 1);
    if (h) {
        PackListToDB(nDbRec[i], h);
        DmReleaseRecord(gListDB, index, true);
    }
}

t1.id          = 3;
t1.type        = kTypeTalk;           // NOTE: the palm OS categories are used to group list 'type's
t1.version     = 1;
t1.count       = 4;                   // NOTE: Zero based index
t1.item[0].type = kTypeAction;
StrCopy( (CharPtr) &(t1.item[0].text), "What can I do for you?");
t1.item[1].type = kTypeName;
StrCopy( (CharPtr) &(t1.item[1].text), "With who?");
t1.item[2].type = kTypePlace;
StrCopy( (CharPtr) &(t1.item[2].text), "Where at?");
t1.item[3].type = kTypeWhen;
StrCopy( (CharPtr) &(t1.item[3].text), "When would you like to do this?");
t1.item[4].type = kTypePlace;
StrCopy( (CharPtr) &(t1.item[4].text), "4 Duh!");
t1.item[5].type = kTypePlace;
StrCopy( (CharPtr) &(t1.item[5].text), "5 Duh!");
t1.item[6].type = kTypePlace;
StrCopy( (CharPtr) &(t1.item[6].text), "6 Duh!");
t1.item[7].type = kTypePlace;
StrCopy( (CharPtr) &(t1.item[7].text), "7 Duh!");
t1.item[8].type = kTypePlace;
StrCopy( (CharPtr) &(t1.item[8].text), "8 Duh!");
t1.item[9].type = kTypePlace;
StrCopy( (CharPtr) &(t1.item[9].text), "9 Duh!");

ii = dmMaxRecordIndex;
h2 = DmNewRecord(gListDB, &ii, 1);
if (MemHandleResize(h2, sizeof(TalkList)) == 0) {

```

Main.c

```

        sP = MemHandleLock(h2);
        i = 0;
        DmWrite(sP, i, &t1, sizeof(TalkList));
        MemHandleUnlock(h2);
    }
    DmReleaseRecord(gListDB, ii, true);
}

/*****
 *
 * FUNCTION:      HandleEsisFieldProcessing
 *
 * DESCRIPTION:   handles text related events in the esistant feild ONLY
 *
 * PARAMETERS:    event to handle
 *
 * RETURNED:      void
 *
 *****/
void HandleEsisFieldProcessing ( EventType *event) {

    if (TxtCharIsPrint(event->data.keyDown.chr)) { // Conversion problem from R5 to R6

        ParseFindSegment (    gListDB, &gUnpkdDbRecd, &matchedItems,
                               &currentPlan, event->data.keyDown.chr);
    }
}

/*****
 *
 * FUNCTION:      CheckOtherNeededDatabases
 *
 * DESCRIPTION:
 *
 * PARAMETERS:
 *
 * RETURNED:      Err error
 *
 *****/
static Err CheckOtherNeededDatabases (void)
{
    // check for '[e-sistant]' category in the address book
    return 0;
}

/*
static CategoriesStruct *GetLockedAppInfo()
{
    UInt  cardNo;
    LocalID dbID;
    LocalID applnfold;

```

Main.c

```

Err      err;

if ((err = DmOpenDatabaseInfo(gProductDB, &dbID, NULL, NULL,
    &cardNo, NULL)) != 0)
    return NULL;
if ((err = DmDatabaseInfo(cardNo, dbID, NULL, NULL, NULL, NULL, NULL,
    NULL, NULL, &applInfoID, NULL, NULL, NULL)) != 0)
    return NULL;
return MemLocalIDToLockedPtr(applInfoID, cardNo);
}
*/
// draw strings at top of rectangle r, but don't overwrite
// right-edge of r

static void DrawCharsToFitWidth(const char *s, RectanglePtr r)
{
    SWord  stringLength = StrLen(s);
    SWord  pixelWidth = r->extent.x;
    Boolean truncate;

    // FntCharsInWidth will update stringLength to the
    // maximum without exceeding the width
    FntCharsInWidth(s, &pixelWidth, &stringLength, &truncate);
    WinDrawChars(s, stringLength, r->topLeft.x, r->topLeft.y);
}

/*****
*
* FUNCTION:      UpdateTalkField
*
* DESCRIPTION:   **This is in the Main.c in order to pickup the global
*                variable 'gListDB' – because of a bug in the compiler
*                the project will not link the globals?? – move it back
*                the into Conversant.c when the bug is fixed.
*
* PARAMETERS:
*
* RETURNED:      Err error
*
*****/

void UpdateTalkField( ULong type)
{
    Int      numOfRecords;
    Handle   rH;
    TalkListPtr r;
    UInt     q, i, recIndex;
    ULong    uniqueID;
    Boolean   gotIt = false;

    if (type == kTypeEnd ) {
        LoadTalkField ("I have everything I need!");
        SysTaskDelay(SysTicksPerSecond());
        LoadTalkField ("Should I take care of it?");
        return;
    }
}

```

Main.c

```
numOfRecords = DmNumRecords(gListDB);

recIndex = 0;
for (i = 0; i < numOfRecords; i++) {
    if (DmSeekRecordInCategory (gListDB, &recIndex, 0, dmSeekForward, dmAllCategories))
        break;
    DmRecordInfo(gListDB, recIndex, NULL, &uniqueID, NULL);
    rH = DmQueryRecord(gListDB, recIndex);
    if (!rH)                                // If we have found a deleted record stop the search.
        break;
    r = (TalkListPtr) MemHandleLock(rH);
    if (r->type == kTypeTalk) {
        for (q = 0; q < r->count; q++) {
            if (r->item[q].type == type) {
                LoadTalkField ((CharPtr) &(r->item[q].text));
                gotIt = true;
                break;
            }
        }
    }
    MemHandleUnlock(rH);
    recIndex++;
}
if (!gotIt)
    LoadTalkField ("I'm confused?!?!");
}
```